Applications of Genetic Algorithms on theoretical fully-autonomous road systems

University of Birmingham

Sam Barrett, 1803086
sjb786@student.bham.ac.uk
Programme: MSci Computer Science
Supervisor: Miriam Backens
Word count: 7462

February 19, 2021

Todo list

check onenote notebook for general ideas about discussion topics etc.

Abstract

Contents

Chapter 1

Introduction

Lay out different tasks & steps of the project, identify subgoals

Move formal goal definitions to Evaluation?

1.1 Goals

The overarching goal of this project is to optimally route traffic in the setting of a fully-autonomous road system. This however, is an aspirational goal with many sub-requirements to be fulfilled before it can come to fruition.

Formally this top-level goal can be said to be to answer the question Problem 1.

We can then split this into more manageable sub-goals. The sub-problem of generating a route through a section of road is defined in Problem 2. This can be further decomposed into the selfish routing of a single agent through a section of road defined in Problem 3

1.1.1 Requirements

The goal of this project was not to produce a production-ready system, but instead, to investigate the plausibility of GAs on the real future possibility of completely autonomous road networks. As such I feel it useful to outline the theoretical requirements of a production grade system. To do this formally I will employ Propositional and Temporal logic.

1. The system should never return a set of routes such that, for any time $t \in T, \forall i \in n, \forall j \in n, j \neq n, I_i(t) = I_j(t)$. I.e. for any time, no routes should inhabit the same point, meaning there are no collisions in the planned routes.

Top-level Goal

Input:

- A road system $\mathcal{R}$ represented as a graph $G = (V, E)$

  Where each edge $e \in E$ is a section of road defined as the space between two vertices $V^1, V^2 \in V$ which is bounded by two functions $b_1(x)$ and $b_2(x)$ and augmented by a set of obstacles O representing infeasible sections of road space

- a set of agents, $A = \{a_1, \ldots, a_n\}$

- a set of vertex pairs representing start and goal points for each agent:

  $\mathcal{V} = \left\{(V_1^1, V_1^2), \ldots, (V_n^1, V_n^2)\right\}, V_{i\in[1\ldots n]}^{j\in[1,2]} \in V$

  where the route for agent a_i is from vertex V_i^1 to V_i^2

Question: What is the set of optimal routes $\mathcal{V}_{optimal}$, where the i^{th} element is defined as a set of linked Bézier curves connecting V_i^1 and V_i^2 through feasible space?

Problem 1

Sub-Goal: Cooperative route planning

Input:

- A start point P_{start} and a goal point P_{goal}
- A section of road as defined in Problem 1
- A knowledge of all routes being planned to be executed concurrently.

Question: What is the optimal route, in the form of a Bézier curve, between these two points s.t. it does not collide with any other agents or pass through any infeasible regions in the road space?

Problem 2

Sub-Goal: Single agent route planning

Input:

- A start point P_{start} and a goal point P_{goal}
- A section of road as defined in Problem 1

Question: What is the optimal route, in the form of a Bézier curve, between these two points s.t. it does not pass through any infeasible regions in the road space?

Problem 3

Sub-Goal: Bézier curve generation

Input:

- A start point P_{start} and a goal point P_{goal}
- A section of road as defined in Problem 1

Question: What is the optimal route, in the form of a Bézier curve, between these two points s.t. it does not pass through any infeasible space?

Problem 4

Chapter 2

Background

2.1 Genetic Algorithms

2.1.1 History

Genetic algorithms are a type of meta-heuristic optimisation technique that employ the same rationale as classical Evolution seen in nature.

Genetic Algorithms can trace their origins back to the late 1960s when they were first proposed by John Holland, though he then referred to them as *Genetic Plans*[1]. Holland went on to write the first book on the subject titled *Adaptation in Natural and Artificial Systems*[3] in 1975. The field did not find much reception with Holland stating in the preface to the 1992 rerun:

> *"When this book was originally published I was very optimistic, envisioning extensive reviews and a kind of 'best seller' in the realm of monographs. Alas! That did not happen."*

However, in the early nineties, Genetic algorithms surged in popularity along with the area of Artificially Intelligent planning as a whole, leading to Holland republishing his book and solidifying his position as the field's founder.

2.1.2 Definition

In a general sense, optimisation techniques work to find the set of parameters $\mathcal{P}$ that minimise an objective function $\mathcal{F}$. Genetic algorithms approach this by representing these sets as individuals in a population, P. Over the course of multiple generations, the best solutions are determined and promoted until termination criteria are met or the maximum number of generations is reached.

[1] This distinction was made to emphasise the *"centrality of computation in defining and implementing the plans"*[3]

As our candidates are essentially a collection of parameters to the function we are trying to optimise, we can extend our metaphor further by mapping each element of a individual to a *gene* in a individual's genome.

The representation we use in a GA is problem specific. Often we have to provide functions to facilitate the mapping between the problem specific set of possible solutions and the encoded genotype space in which we optimise. The most basic representation being a string of binary numbers.

An individual's characteristics and genetic information is normally encapsulated within the Phenotype. Here not only the genetic information of the Genotype but also additional information, such as fitness, is stored in order to prevent it from being re-calculated as often.

Genetic algorithms are both *probabilistically optimal* and *probabilistically complete*[1] meaning that: given infinite time, not only will the algorithm find *a* solution, (if one exists), it will find **the** optimal solution from the set of all possible solutions, $\mathcal{P}^*$.

Algorithm 1: Modern Generic Genetic Algorithm

Result: Best Solution, $p_{\texttt{best}}$
Generate initial population, P_0 of size n;
Evaluate fitness of each individual in P_0, $\{F(p_{0,1}, \ldots, p_{0,n})\}$;
while *termination criteria are not met* **do**
  Selection: Select individuals from P_t based on their fitness;
  Variation: Apply variation operators to parents from P_t to produce offspring;
  Evaluation: Evaluate the fitness of the newly bred individuals;
  Reproduction: Generate a new population P_{t+1} using individuals from P_t as well as the newly bred candidates.;
  t++
end
return $p_{\texttt{best}}$

As you can see from Figure 2.1 and Algorithm 1 the overall shape of GAs has not changed substantially over the course of the past 50 years. Being comprised of a series of operations that a starting population is piped through until criteria are met.

2.1.3 Genetic Operators

In the following section I will outline the various genetic operations that take place in a GA, they can be seen in Algorithm 1. Any operators used and analysed in Chapter 4 will have more detailed explanations of their process.

Expand definitions of operators used in implementation section

Selection

The selection procedure is the process by which the next generation of individuals is created from the current population. Individuals are selected

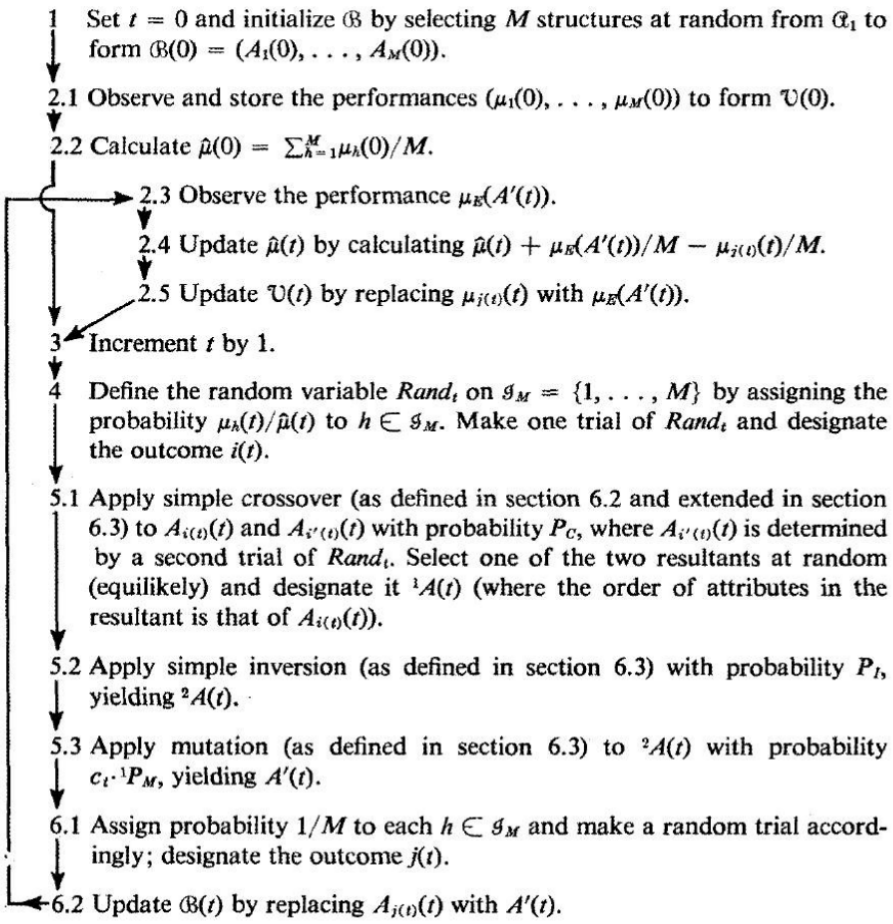

Figure 2.1: GA algorithm outlined in Holland's Original Book[3]

relative to their fitness as determined by the Objective (fitness) function $\mathcal{F}$.

Some methods select only the best solutions by fitness. Others employ a more stochastic approach, such as roulette wheel selection, to increase diversity and reduce complexity.

Fitness Proportional Selection Fitness proportional or Roulette wheel selection is a popular selection operator. It uses fitness to assign selection probability to each individual in a population.

The probability of selection for an individual i with fitness $\mathcal{F}(i)$ can be expressed mathematically as:

$$p_i = \frac{\mathcal{F}(i)}{\sum_{j=1}^{N} \mathcal{F}(j)} \tag{2.1}$$

Where N is the size of the population P

This is a simple approach but performs well and has very little performance overhead.

Ranked Selection Ranked selection is an alternative selection method, it works on the assumption that the individuals in your population are closely grouped in fitness. The main difference of this method when compared to fitness proportional selection is that it ranks individuals based on relative fitness rather than absolute fitness.

The procedure can be written mathematically as:

Given a population P of size n ordered by fitness. We select the top γ-ranked individual, x_γ with a probability $p(\gamma)$. Where γ is the rank and $p(\gamma)$ is the ranking function.

The ranking function can take different forms including both linear and exponential ranking.

extend list to include all functions used

Linear ranking is defined as:

$$p(\gamma) = \frac{\alpha + (\beta - \alpha) \cdot \frac{\gamma}{n-1}}{n} \tag{2.2}$$

Where:

- $\sum_{\gamma=0}^{n} p(\gamma) = 1$
- $\alpha + \beta = 2$
- $1 \leq \beta \leq 2$

Variation

Variation in a GA is the process of altering the genome individuals to further explore the search space via stochastic local search. We perform this using two distinct sub-operations: Mutation and Crossover. Here we can view crossover as the *breeding* process and mutation as resembling the natural tendency for DNA to mutate over the course of generations.

Mutation In mutation we alter each gene with a set probability p_m known as the *mutation rate*. A standard value for a mutation rate is $\frac{1}{L}$ but it can fall anywhere in the range $p_m \in [\frac{1}{L}, \frac{1}{2}]$ where L is the length of the genome.

A low value for p_m a new individual which can be shown to be *close* to it's parents in the search space relative to their *Hamming distance*[2] if using a Binary coded GA. In real-coded GAs they can be shown to be close by the Euclidean distance between them.

In binary coded GAs we alter a given gene by *flipping* it's value. In real-coded GAs mutation operators include:

- Uniform Mutation
- Non-Uniform Mutation
- Gaussian Mutation

Uniform Mutation In uniform mutation, we select a parent p at random and replace a randomly selected gene $c_i \in p$ with a uniformly random number $c_i' \in [u_i, v_i]$ where u_i and v_i are set bounds.

Gaussian Mutation In Gaussian mutation, we select a parent p at random and randomly select a gene $c_i \in p$. We replace c_i with a new value c_i' which is calculated as follows:

$$c_i' = \min(\max(\mathcal{N}(c_i, \sigma_i), u_i), v_i) \tag{2.3}$$

Where $\mathcal{N}(c_i, \sigma_i)$ is a Gaussian distribution with a standard deviation σ_i and a mean of c_i. σ_i may depend on the length of the interval bound, $l_i = v_i - u_i$, typically (and in my implementation) $\sigma_i = \frac{1}{10} l_i$

Crossover

Crossover is a binary operation, taking two randomly selected *parents* from the population P_t with a probability $p_c \in [0, 1]$

There are two major forms of crossover for binary coded GAs: $n > 1$ point crossover and Uniform crossover.

[2]Hamming Distance: The metric for comparing two binary data strings. The Hamming distance between two strings is the number of bit position in which they differ.

For real-coded GAs you instead have a selection of Crossover operators including:

- Flat crossover
- Simple crossover
- Whole arithmetic crossover
- Local arithmetic crossover
- Single arithmetic crossover
- BLX-α crossover

Simple (one point) Crossover For simple crossover, randomly select a crossover point, $i \in \{1, \ldots, n\}$. All values before this point are swapped between the two parents. For 2 parents, $p_1 = \left\{x_1^{[1]}, \ldots, x_n^{[1]}\right\}$ and $p_2 = \left\{x_1^{[2]}, \ldots, x_n^{[2]}\right\}$ this can be represented as:

$$p_1' = \left\{x_1^{[1]}, x_2^{[1]}, \ldots, x_i^{[1]}, x_{i+1}^{[2]}, \ldots, x_n^{[2]}\right\} \tag{2.4}$$

$$p_2' = \left\{x_1^{[2]}, x_2^{[2]}, \ldots, x_i^{[2]}, x_{i+1}^{[1]}, \ldots, x_n^{[1]}\right\} \tag{2.5}$$

Evaluation

After developing potential new individuals, the fitness of these new individuals is calculated using the objective function, $\mathcal{F}$

Reproduction

Here the next generation of individuals is constructed. There are multiple potential methods employed here with the simplest being to just replace the lest fit individuals with additional copies of the fitter individuals, be they pre-existing or newly generated.

In this stage various heuristics or alternative strategies can be implemented to speed up or slow down the convergence rate. Whilst their fitness may be low, having a diverse population allows for more of the search space to be explored. If the algorithm converges too quickly it may get *stuck* in local minima.

2.1.4 Bézier Curves

In my implementation, I utilise Bézier curves to encode complex route arcs between a series of points. Here I will briefly outline their construction and mathematical basis.

Bézier curves were popularised by and named after French auto-body[3] designer Pierre Bézier in the 1960s and are commonly found in computer graphics today. They are parametric curves made up of a series of *control points* which contort the shape of a line to produce a curve of nearly any shape. Although their mathematical basis was established in 1912 by Sergei Bernstein in the form of Bernstein polynomials[4].

A Bézier curve is said to have a degree of $n-1$ where n is the number of control points including the start and end point of the curve.

Formal Definition

Bézier curves can be simply explicitly defined for degrees of 1 through 4, however, the general case of n points is more useful to us. In their general form they can be defined recursively or explicitly. I implement the recursive version as it is much more readable for a programming language that supports recursion.

They are defined recursively as follows:

$$\mathbf{B}_{P_0}(t) = P_0 \tag{2.6}$$

$$\mathbf{B}_{P_0,P_1,\ldots,P_n}(t) = (1-t)\mathbf{B}_{P_0,P_1,\ldots,P_{n-1}}(t) + t\mathbf{B}_{P_1,P_2,\ldots,P_n}(t) \tag{2.7}$$

Where the parameter t is in the range of $[0,1]$. Therefore, the smoothness of the curve is determined by the granularity of the parameter when realising the curve.

Subdivision of n-degree Bézier curves

Subdivision of Bézier curves is a key stage in the process of finding any intersections between two Bézier curves. The process of subdivision is performed using De Casteljau's algorithm. This algorithm was developed by Paul de Casteljau while working at Citroen in 1959 to work with the family of curves that would later be formalised into Bézier curves by Pierre Bézier[4].

It is a simple procedure, given a Bézier curve B, it can be split at any arbitrary point $t \in [0,1]$ into two curves with control points of the form:

[3] so I feel it is quite fitting that they find use in 21^{st} century automotive problems

[4] Bézier curves have also been called de Casteljau curves as much of his work preceded that of Pierre Bézier's

$$B^{(1)} = B_{P_0}(t), B_{P_0,P_1}(t), \ldots, B_{P_0,\ldots,P_n}(t) \tag{2.8}$$
$$B^{(2)} = B_{P_n}(1-t), B_{P_n,P_{n-1}}(1-t), \ldots, B_{P_n,\ldots,P_0}(1-t) \tag{2.9}$$

Should I change this superscript notation? subscript is being used to denote which control points are being considered.

2.2 Fully Autonomous Road Networks

Fully autonomous road networks are by no means a novel concept. They have appeared in fiction since the mid 20th century and have been a real possibility for the past decade. In a fully autonomous road networks *(FARN)*s, all vehicles are self-operating with their routing either being determined on a vehicle-by-vehicle *selfish* basis or by a larger system managing routes for all nearby agents.

In the latter system protocols that promote the net increase in efficiency can be implemented. However, this sort of system will require a form of government mandate as a universal routing protocol would need to be established and implemented by all auto manufacturers; this is no small undertaking.

Chapter 3

Literature Review

3.1 Genetic algorithms for route generation

Classical Genetic algorithms have seen research and applications in a number of fields over the past 40 years ranging from Heart disease diagnosis[5] to predicting the strength of concrete under various conditions[6].

There is a substantial and growing body of research specifically looking at the applications of GAs in route planning for autonomous agents, however, many are focusing on either abstract robotic agent planning in discrete search spaces or failing to take into account the wider environment of multiple agents.

3.1.1 Genetic algorithms for cooperative route generation

Cai & Peng - Cooperative Coevolutionary Adaptive Genetic Algorithm in Path Planning of Cooperative Multi-Mobile Robot Systems[2]
In this 2001 paper, Cai & Peng give an example of a Cooperative Coevolutionary Adaptive Genetic Algorithm (CCAGA) for planning routes for multiple autonomous agents.

Their approach features a discrete grid-based search space featuring rectangular obstacles which the routes must avoid. Their chromosomes are encoded using real-values representing the x and y coordinates of the search grid.

Their fitness function has two stages, in the first generation the fitness of a candidate solution is purely determined by the global *optimality* of the route. In all subsequent generations the fitness also takes into account the best routes from each of the other sub-populations.

They conclude that their approach yields good robustness and convergence as well as being suitable for parallel execution, allowing for systems employing this GA approach to run very efficiently.

make reference to this method being altered for use in my solution

Raul Kala - Optimization Based Planning 2016[7] In his book, Kala proposes a GA as an embedded component of a larger planning system which also relies on Djiksra's algorithm for macro-level route planning along with the classical autonomous vehicle real-time collision avoidance toolbox (radarr, sonarr, etc.). In order to achieve a system capable of planning for multiple agents concurrently, Kala proposed the use of these real-time collision avoidance tools along with embedding *traffic rules* as heuristics into the Djiksta-based component. These traffic rules included "driving on the left" and "overtaking on the right" etc. This proposed system, while similar to mine, differs in scope. I propose a system for planning routes for a set of autonomous agents within a fully-autonomous road system, meaning agents will not (in theory) interact or encounter any other vehicles which are not a part of the planning network.

Talk about Kala's road space coordinate system & use of Bézier curves, move to Bézier subsection?

Cruz-Piris et al. - Automated Optimsization of Intersections Using a Genetic Algorithm[8] A paper published by Cruz-Piris et al. investigates the applications of Genetic Algorithms on routing traffic through busy intersections. Their proposed system is shown to have improvements in traffic throughput of anywhere from 9% to 36%.

Their approach involved several assumptions, many of which my research shares. Namely, they assumed vehicles do not stop at any point and that their speed remains constant at all points in time.

They modelled intersections and associated traffic flows using (Traffic) Cellular Automata (TCA) as proposed by Maerivoet and De Moor[9]. This approach takes advantage of the fairly uniform and regular structure of large intersections, and although they work on modelling irregularly shaped intersections, this TCA model begins to fall apart when the size of each cell cannot contain a vehicle and requisite surrounding space or if input/output lanes do not lie in a regular grid pattern.

Pare this down?

To circumvent the latter issue, they propose creating "virtual lanes based on the input and output paths", though they do not elaborate as to how these lanes are created. With regards to the grid cell size, they also have to ensure that these new *virtual lanes* fit into the grid along with vehicles, again, they do not specify a method for selecting cell size, only providing requirements and examples. This leads me to believe that these were calculated by hand through trial-and-error, not a viable method when you consider the number of unique intersections across the world or even just the Continental United States!

They go on to define their GA approach to routing agents between the set of input and output points. They use a binary coded GA, this is possible due to their use of discretised search spaces. Their chromosomes are codified as a long bistring where each bit represents the presence of absence of a vehicle at a given grid square for each of the n *arms* of the junction. This is

a relatively efficient encoding, with checks for vehicle presence being trivial.

remove this para?

This approach seems to work relatively well for throughput optimisation problems such as junctions but does not scale well for less structured scenarios where the origins and goals cannot be known beforehand and are not necessarily the same for all subsequent runs.

3.2 Bézier curves for route generation

Chen et al. - Quartic Bézier Curve based Trajectory Generation for Autonomous Vehicles with Curvature and Velocity Constraints In this 2014 paper, Chen et. al research the efficacy of quartic (4-degree) Bézier curves for planning continuous routes between points for autonomous vehicles. In order to find an optimal solution they employ sequential quadratic programming.

In their approach Chen et al. also take into account the orientation, change in orientation and velocity of the routes generated in order to assure all generated routes are *safe* for occupied vehicles to follow.

This approach appears to be extremely computationally efficient on the toy examples provided in the paper, on relatively low performance hardware 560 iterations can be computed in around 500ms!

There is no mention, however, to a cooperative element. Explicitly solving quadratic programming tasks is known to be **NP**-complete as shown by Vavasis in 1990[10] and adding onto that the requirement that all solutions interact nicely will only increase the real-world runtime of any solution. A real-world scenario with hundreds of vehicles may suffer from an incredibly high runtime, hardware requirement or both.

is this even correct to say?

3.3 Fully Autonomous Road Networks

*FARN*s have the possibility of improving travel in a number of different areas.

They have the potential of greatly reducing travel times by efficiently routing all vehicles with the goal of reducing the net travel time. Such a system would also be able to respond much faster than human drivers, allowing for large increases in permissible vehicle velocities thus, further increasing efficiency.

By extension of their higher efficiency, *FARN*s will also have a marked effect on vehicular energy consumption. The potential effects are summarised nicely by Vahidi and Sciarretta[11] where they show the use of vehicle automation could cause anywhere from a halving of "energy use and greenhouse gas emission" in an "optimistic scenario" to doubling them depending on the

effects that present themselves. The increase in highway speed whilst increasing travel efficiency is predicted by Brown et al.[12] and Wadud et al.[13] as increasing energy use by anywhere from 5% to 30%

Chapter 4

Approach

4.1 Approach

The general shape of a GA as seen in Algorithm 1 is the same for almost all problems.

As such, I first had to implement a method to generate an initial population. In order to do this, I first had to define my programmatic representations of the Phenotypes and genotypes of my individuals. I decided to base my genotypic structure on the that described by Kala[1].

In this approach, the genotype is abstractly represented as a Bézier curve. This is more concretely translated into a real-valued string, interleaving the x and y coordinates of each control point.

Initially my Phenotype contained no additional information. However, I still created the distinction to allow for additional properties to be encapsulated during development.

The foundation of my Fitness function is the length of each route, i.e. the length of a given Bézier curve. This is not a trivial calculation and the best approximation was found to be:

$$L = \frac{(2L_c + (n-1)L_p)}{n+1} \tag{4.1}$$

Where L_c is chord length and L_p is polygon length. This was proven by Jens Gravesen[14] in 1997

This is just a placeholder so I don't forget to include this

4.1.1 Co-evolution of routes for a set of agents

So far we have only concerned ourselves with planning a route through a road-space for a single agent. However, in the real world, roads are seldom occupied by a single vehicle and as such we must consider how to efficiently plan a set of routes for a set of agents between between a set of coordinate pairs, such that, our agents do not collide for any time t.

There are different approaches we can take to this problem.

refer to lit rev sections discussing approaches by Kala[1] and Cai & Peng [2]

My initial approach to a collaborative planning system was somewhat inefficient. The system operated by planning routes for each agent sequentially, using a growing *context* to keep track of the routes that had already been finalised. The first agent to be planned was not concerned with avoiding collisions, as, as far as it was concerned, there were no other agents in the system. Each subsequent agent checked if any of its candidate solutions intersected with and of the routes planned before, a very high penalty was applied to any routes with collisions, equivalent to the distance

check this

the routes collided for.

This system was very inefficient with the final routes requiring a large number of checks to be carried out by my `bezInt` function, which itself suffered from

verify this

exponential time (and space) complexity.

4.2 Implementation

My for my initial implementation, I focused on simply getting a working system. I therefore, started by implementing the simplest form of each genetic operator. Namely, fitness proportional selection (aka. roulette wheel selection

check this

), simple crossover and uniform mutation. (2.1.3)

4.2.1 Language Choice

I have chosen to implement my approach using the Julia language project[15].

Julia is a relatively new language first developed in 2012 by Jeff Bezanson, Stefan Karpinski and Viral Shah. It is a multi-paradigm language allowing for functional, object oriented and meta programming approaches to problems. I will mainly be using it for it's functional and OO capabilities.

Julia operates using multiple dispatch similar to languages such as Haskell. It interoperates with C and Fortran codebases without the need of middleman bloat. This fact allows it to utilise the extensive high performance C libraries for floating point operations. Julia is eagerly evaluated, uses a Just in time compiler and has a garbage collector.

Julia features a syntax similar to both Python and Matlab with performance on par with C. As I am already very familiar with python and have

studied functional programming in a number of modules; I found this language very quick and intuitive to learn and the resulting code to be clean, idiomatic and fast.

A real-world deployment of a system based on my research would undoubtedly be required to run on small, relatively low performance, embedded systems and as such Julia may not be appropriate here. A language such as C or Rust may be used instead.

Julia also has distributed computation facilities. This sort of functionality could be extremely useful in a system such as mine as it could allow for computation to be spread across the agents themselves, removing the need for a central planning center which could be a single point of failure.

4.3 Results

My results were achieved by

Chapter 5

Evaluation

5.0.1 Bézier Curves

expand on this section, talk about issues of finding intersection, possible GPU applications

Bézier curves have been utlised in this project to encode and represent the route of a vehicle. As mentioned in Section 2.1.4, there are many reasons I originally selected them for this task. However, over the course of implementation and testing, a number of downsides have been presented.

5.0.2 Cooperative Planning

My solution to the problem of planning n non-colliding routes for n agents was so wrap my existing `GA` function in a cooperative *layer*. This cooperative layer relied on a function for detecting collisions which had extremely high overhead, at one point causing around 50x slowdown in the running time of the function. Detecting intersections between two Bézier curves is a non-trivial task with the best methods taking the same approach of recursive subdivision that I utilised.

Chapter 6

Conclusion

Bibliography

[1] Rahul Kala. *On-Road Intelligent Vehicles: Motion Planning for Intelligent Transportation Systems / Rahul Kala.* Butterworth-Heinemann is an imprint of Elsevier, Kidlington, Oxford, UK, 2016.

[2] Zixing Cai and Zhihong Peng. Cooperative Coevolutionary Adaptive Genetic Algorithm in Path Planning of Cooperative Multi-Mobile Robot Systems. *Journal of Intelligent and Robotic Systems*, 33(1):61–71, January 2002.

[3] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT Press, Cambridge, UNITED STATES, 1992.

[4] SN Bernstein. On the best approximation of continuous functions by polynomials of a given degree. *Comm. Soc. Math. Kharkow, Ser*, 2(13):49–194, 1912.

[5] G. Thippa Reddy, M. Praveen Kumar Reddy, Kuruva Lakshmanna, Dharmendra Singh Rajput, Rajesh Kaluri, and Gautam Srivastava. Hybrid genetic algorithm and a fuzzy logic classifier for heart disease diagnosis. *Evolutionary Intelligence*, 13(2):185–196, June 2020.

[6] Mahdi Shariati, Mohammad Saeed Mafipour, Peyman Mehrabi, Masoud Ahmadi, Karzan Wakil, Nguyen Thoi Trung, and Ali Toghroli. Prediction of concrete strength in presence of furnace slag and fly ash using Hybrid ANN-GA (Artificial Neural Network-Genetic Algorithm). *Smart Structures and Systems*, 25(2):183–195, 2020.

[7] Rahul Kala. Optimization-Based Planning. In *On-Road Intelligent Vehicles*, pages 109–150. Elsevier, 2016.

[8] Luis Cruz-Piris, Miguel A. Lopez-Carmona, and Ivan Marsa-Maestre. Automated Optimization of Intersections Using a Genetic Algorithm. *IEEE Access*, 7:15452–15468, 2019.

[9] Sven Maerivoet and Bart De Moor. Cellular automata models of road traffic. *Physics Reports*, 419(1):1–64, November 2005.

[10] Stephen A. Vavasis. Quadratic programming is in NP. *Information Processing Letters*, 36(2):73–77, 1990.

[11] Ardalan Vahidi and Antonio Sciarretta. Energy saving potentials of connected and automated vehicles. *Transportation Research Part C: Emerging Technologies*, 95:822–843, October 2018.

[12] Austin Brown, Jeffrey Gonder, and Brittany Repac. An Analysis of Possible Energy Impacts of Automated Vehicles. In Gereon Meyer and Sven Beiker, editors, *Road Vehicle Automation*, Lecture Notes in Mobility, pages 137–153. Springer International Publishing, Cham, 2014.

[13] Zia Wadud, Don MacKenzie, and Paul Leiby. Help or hindrance? The travel, energy and carbon impacts of highly automated vehicles. *Transportation Research Part A: Policy and Practice*, 86:1–18, April 2016.

[14] Jens Gravesen. Adaptive subdivision and the length and energy of Bézier curves. *Computational Geometry*, 8(1):13–31, June 1997.

[15] The Julia Programming Language. https://julialang.org/.

Appendices